The `phylo4` S4 classes and methods

Ben Bolker & Peter Cowan

August 18, 2009

Contents

1 Introduction

This document describes the new `phylo4` S4 classes and methods, which are intended to provide a unifying standard for the representation of phylogenetic trees and comparative data in R. The `phylobase` package was developed to help both end users and package developers by providing a common suite of tools likely to be shared by all packages designed for phylogenetic analysis, facilities for data and tree manipulation, and standardization of formats.

This standardization will benefit *end-users* by making it easier to move data and compare analyses across packages, and to keep comparative data synchronized with phylogenetic trees. Users will also benefit from a repository of functions for tree manipulation, for example tools for including or excluding subtrees (and associated phenotypic data) or improved tree and data plotting facilities. `phylobase` will benefit *developers* by freeing them to put their programming effort into developing new methods rather than into re-coding base tools. We (the `phylobase` developers) hope `phylobase` will also facilitate code validation by providing a repository for benchmark tests, and more generally that it will help catalyze community development of comparative methods in R.

A more abstract motivation for developing `phylobase` was to improve data checking and abstraction of the tree data formats. `phylobase` can check that data and trees are associated in the proper fashion, and protects users and developers from accidently reordering one, but not the other. It also seeks to abstract the data format so that commonly used information (for example, branch length information or the ancestor of a particular node) can be accessed without knowledge of the underlying data structure (i.e., whether the tree is stored as a matrix, or a list, or a parenthesis-based format). This is achieved through generic `phylobase` functions which which retrieve the relevant information from the data structures. The benefits of such abstraction are multiple: (1) *easier access to the relevant information* via a simple function call (this frees both users and developers from learning details of complex data structures), (2) *freedom to optimize data structures in the future without breaking code.* Having the generic functions in place to "translate" between the data structures and the rest of the program code allows program and data structure development to proceed somewhat independently. The alternative is code written for specific data structures, in which modifications to the data structure requires rewriting the entire package code (often exacting too high a price, which results in the persistence of less-optimal data structures). (3) *providing broader access to the range of tools in* `phylobase`. Developers of specific packages can use these new tools based on S4 objects without knowing the details of S4 programming.

The base `phylo4` class is modeled on the the `phylo` class in `ape`. `phylo4d` and `multiphylo4` extend the `phylo4` class to include data or multiple trees respectively. In addition to describing the classes and methods, this vignette gives examples of how they might be used.

2 Package overview

The phylobase package currently implements the following functions and data structures:

- Data structures for storing a single tree and multiple trees: `phylo4` and `multiPhylo4`?

- A data structure for storing a tree with associated tip and node data: `phylo4d`
- A data structure for storing multiple trees with one set of tip data: `multiPhylo4d`
- Functions for reading nexus files into the above data structures
- Functions for converting between the above data structures and `ape phylo` objects as well as `ade4 phylog` objects
- Functions for editing trees and data (i.e., subsetting and replacing)
- Functions for plotting trees and trees with data

3 Using the S4 help system

The `S4` help system works similarly to the `S3` help system with some small differences relating to how `S4` methods are written. The `plot()` function is a good example. When we type `?plot` we are provided the help for the default plotting function which expects `x` and `y`. `R` also provides a way to smartly dispatch the right type of plotting function. In the case of an `ape phylo` object (a `S3` class object) `R` evaluates the class of the object and finds the correct functions, so the following works correctly.

```
> library(ape)
> rand_tree <- rcoal(10)
> plot(rand_tree)
```

However, typing `?plot` still takes us to the default `plot` help. We have to type `plot.phylo` to find what we are looking for. This is because `S3` generics are simply functions with a dot and the class name added.

The `S4` generic system is too complicated to describe here, but doesn't include the same dot notation. As a result `?plot.phylo4` doesn't work, `R` does, however, find the right plotting function.

```
> library(phylobase)
> rand_p4_tree <- as(rand_tree, "phylo4")
> plot(rand_p4_tree)
```

All fine and good, but how to we find out about all the great features of the `phylobase` plotting function? `R` has two nifty ways to find it, the first is to simply put a question mark in front of the whole call:

```
> ?plot(rand_p4_tree)
```

`R` looks at the class of the `rand_p4_tree` object and takes us to the correct help file (note: this only works with `S4` objects). The second ways is handy if you already know the class of your object, or want to compare to generics for different classes:

```
> method?plot("phylo4")
```

More information about how S4 documentation works can be found in the methods package, by running the following command.

```
> help("Documentation", package = "methods")
```

4 Trees without data

You can start with a tree — an object of class `phylo` from the `ape` package (e.g., read in using the `read.tree()` or `read.nexus()` functions), and convert it to a `phylo4` object.

For example, load the raw *Geospiza* data:

```
> library(phylobase)
> data(geospiza_raw)
> names(geospiza_raw)
```

```
[1] "tree" "data"
```

Convert the S3 tree to a S4 `phylo4` object using the `as()` function:

```
> (g1 <- as(geospiza_raw$tree, "phylo4"))
```

```
          label node ancestor edge.length node.type
15         <NA>   15       NA          NA      root
16         <NA>   16       15     0.29744  internal
17         <NA>   17       16     0.04924  internal
18         <NA>   18       17     0.06859  internal
19         <NA>   19       18     0.13404  internal
20         <NA>   20       19     0.10346  internal
21         <NA>   21       20     0.03550  internal
22         <NA>   22       21     0.00917  internal
23         <NA>   23       22     0.07333  internal
24         <NA>   24       23     0.05500  internal
25         <NA>   25       19     0.24479  internal
26         <NA>   26       25     0.05167  internal
27         <NA>   27       26     0.01500  internal
1    fuliginosa    1       24     0.05500       tip
2        fortis    2       24     0.05500       tip
3  magnirostris    3       23     0.11000       tip
4   conirostris    4       22     0.18333       tip
5      scandens    5       21     0.19250       tip
6    difficilis    6       20     0.22800       tip
7       pallida    7       25     0.08667       tip
8      parvulus    8       27     0.02000       tip
9    psittacula    9       27     0.02000       tip
```

```
10        pauper   10       26      0.03500       tip
11    Platyspiza   11       18      0.46550       tip
12         fusca   12       17      0.53409       tip
13 Pinaroloxias   13       16      0.58333       tip
14      olivacea   14       15      0.88077       tip
```

The (internal) nodes appear with labels `<NA>` because they are not defined:

```
> nodeLabels(g1)

character(0)
```

(this means "a character vector of length 0"; you can also retrieve the node labels with `labels(g1,"internal")`).

A simple way to assign the node numbers as labels (useful for various checks) is

```
> nodeLabels(g1) <- paste("N", nodeId(g1), sep = "")
> head(g1, 5)

   label node ancestor edge.length node.type
15   N15   15       NA          NA      root
16   N16   16       15     0.29744  internal
17   N17   17       16     0.04924  internal
18   N18   18       17     0.06859  internal
19   N19   19       18     0.13404  internal
```

The `summary` method gives a little extra information, including information on branch lengths:

```
> summary(g1)

 Phylogenetic tree : g1

 Number of tips    : 14
 Number of nodes   : 13
 Branch lengths:
        mean         : 0.1764008
        variance     : 0.04624379
        distribution :
   Min. 1st Qu.  Median 3rd Qu.    Max.    NA's
0.00917 0.04985 0.08000 0.21910 0.88080 1.00000
```

Print tip labels:

```
> tipLabels(g1)
```

```
            1               2               3               4               5 
  "fuliginosa"        "fortis" "magnirostris"  "conirostris"     "scandens" 
            6               7               8               9              10 
  "difficilis"       "pallida"     "parvulus"   "psittacula"       "pauper" 
           11              12              13              14 
  "Platyspiza"         "fusca" "Pinaroloxias"     "olivacea" 
```

(`labels(g1)` or `labels(g1,"tip")` would also work.)

Print node numbers (in edge matrix order):

```
> nodeId(g1, type = "allnode")
```

```
 [1] 15 16 17 18 19 20 21 22 23 24 25 26 27  1  2  3  4  5  6  7  8  9 10 11 12
[26] 13 14
```

Print edge labels (also empty in this case — therefore all `NA`):

```
> edgeLabels(g1)
```

```
15-16 16-17 17-18 18-19 19-20 20-21 21-22 22-23 23-24  24-1  24-2  23-3  22-4 
   NA    NA    NA    NA    NA    NA    NA    NA    NA    NA    NA    NA    NA 
 21-5 NA-15  20-6 19-25  25-7 25-26 26-27  27-8  27-9 26-10 18-11 17-12 16-13 
   NA    NA    NA    NA    NA    NA    NA    NA    NA    NA    NA    NA    NA 
15-14 
   NA 
```

Is it rooted?

```
> isRooted(g1)
```

```
[1] TRUE
```

Which node is the root?

```
> rootNode(g1)
```

```
[1] 15
```

Does it contain any polytomies?

```
> hasPoly(g1)
```

```
[1] FALSE
```

Does it have branch lengths?

```
> hasEdgeLength(g1)
```

```
[1] TRUE
```

You can modify labels and other aspects of the tree — for example, to convert all the labels to lower case:

```
> tipLabels(g1) <- tolower(tipLabels(g1))
```

You could also modify selected labels, e.g. to modify the labels in positions 11 and 13 (which happen to be the only labels with uppercase letters):

```
> tipLabels(g1)[c(11, 13)] <- c("platyspiza", "pinaroloxias")
```

5 Trees with data

The `phylo4d` class matches trees with data, or combines them with a data frame to make a `phylo4d` (tree-with-data) object.

Now we'll take the *Geospiza* data from `geospiza_raw$data` and merge it with the tree. However, since *G. olivacea* is included in the tree but not in the data set, we will initially run into some trouble:

```
> g2 <- phylo4d(g1, geospiza_raw$data)
```

```
Error in switch(missing.data, warn = warning(msg), fail = stop(msg)) :
  The following nodes are not found in the dataset:  platyspiza, pinaroloxias, olivacea
```

We have two problems — the first is that we forgot to lowercase the labels on the data to match the tip labels:

```
> gdata <- geospiza_raw$data
> row.names(gdata) <- tolower(row.names(gdata))
```

To deal with the second problem (missing data for *G. olivacea*), we have a few choices. The easiest is to use `missing.tip.data="OK"` to allow R to create the new object:

```
> g2 <- phylo4d(g1, gdata, missing.data="warn")
```

(setting `missing.data` to `"warn"` would create the new object but print a warning).

Another way to deal with this would be to use `prune()` to drop the offending tip from the tree first:

```
> g1B <- prune(g1, "olivacea")
> phylo4d(g1B, gdata)
```

You can summarize the new object:

```
> summary(g2)
```

```
 Phylogenetic tree : as(x, "phylo4")

 Number of tips    : 14
 Number of nodes   : 13
 Branch lengths:
        mean         : 0.1764008
        variance     : 0.04624379
        distribution :
   Min. 1st Qu.  Median 3rd Qu.    Max.    NA's
0.00917 0.04985 0.08000 0.21910 0.88080 1.00000
```

```
Comparative data:

Tips: data.frame with 14 taxa and 5 variable(s)

     wingL          tarsusL         culmenL          beakD
 Min.   :3.975   Min.   :2.807   Min.   :1.974   Min.   :1.191
 1st Qu.:4.189   1st Qu.:2.929   1st Qu.:2.187   1st Qu.:1.941
 Median :4.235   Median :2.980   Median :2.311   Median :2.073
 Mean   :4.236   Mean   :2.991   Mean   :2.333   Mean   :2.083
 3rd Qu.:4.265   3rd Qu.:3.039   3rd Qu.:2.430   3rd Qu.:2.347
 Max.   :4.420   Max.   :3.271   Max.   :2.725   Max.   :2.824
 NA's   :1.000   NA's   :1.000   NA's   :1.000   NA's   :1.000
     gonysW
 Min.   :1.401
 1st Qu.:1.845
 Median :1.962
 Mean   :2.014
 3rd Qu.:2.222
 Max.   :2.676
 NA's   :1.000

Object contains no node data.
```

Or use `tdata()` to extract the data (i.e., `tdata(g2)`). By default, `tdata()` will retrieve tip data, but you can also get internal node data only (`tdata(tree, "internal")`) or — if the tip and node data have the same format — all the data combined (`tdata(tree, "allnode")`).

If you want to plot the data (e.g. for checking the input), `plot(tdata(g2))` will create the default plot for the data — in this case, since it is a data frame [**this may change in future versions but should remain transparent**] this will be a `pairs` plot of the data.

6 Subsetting

The `subset` command offers a variety of ways of extracting portions of a `phylo4` or `phylo4d` tree, keeping any tip/node data consistent.

tips.include give a vector of tips (names or numbers) to retain

tips.exclude give a vector of tips (names or numbers) to drop

mrca give a vector of node or tip names or numbers; extract the clade containing these taxa

node.subtree give a node (name or number); extract the subtree starting from this node

Different ways to extract the *fuliginosa-scandens* clade:

```
> subset(g2, tips.include = c("fuliginosa", "fortis", "magnirostris",
+     "conirostris", "scandens"))
> subset(g2, node.subtree = 21)
> subset(g2, mrca = c("scandens", "fortis"))
```

One could drop the clade by doing

```
> subset(g2, tips.exclude = c("fuliginosa", "fortis", "magnirostris",
+     "conirostris", "scandens"))
> subset(g2, tips.exclude = names(descendants(g2, MRCA(g2, c("difficilis",
+     "fortis")))))
```

7 Tree-walking

`getnodes`, `children`, `parent`, `descendants`, `ancestors`, `siblings`, `MRCA` ...
generally take a `phylo4` object, a node (specified by number or name) and return a named vector of node numbers.

8 multiPhylo classes

Fix me!

9 Examples

9.1 Constructing a Brownian motion trait simulator

This section will describe two (?) ways of constructing a simulator that generates trait values for extant species (tips) given a tree with branch lengths, assuming a model of Brownian motion.

9.1.1 the easy way

We can use `as(tree,"phylo4vcov")` to coerce the tree into a variance-covariance matrix form, and then use `mvrnorm` from the `MASS` package to generate a set of multivariate normally distributed values for the tips. (A benefit of this approach is that we can very quickly generate a very large number of replicates.) This example illustrates a common feature of working with `phylobase` — combining tools from several different packages to operate on phylogenetic trees with data.

We start with a randomly generated tree using `rcoal()` from `ape` to generate the tree topology and branch lengths:

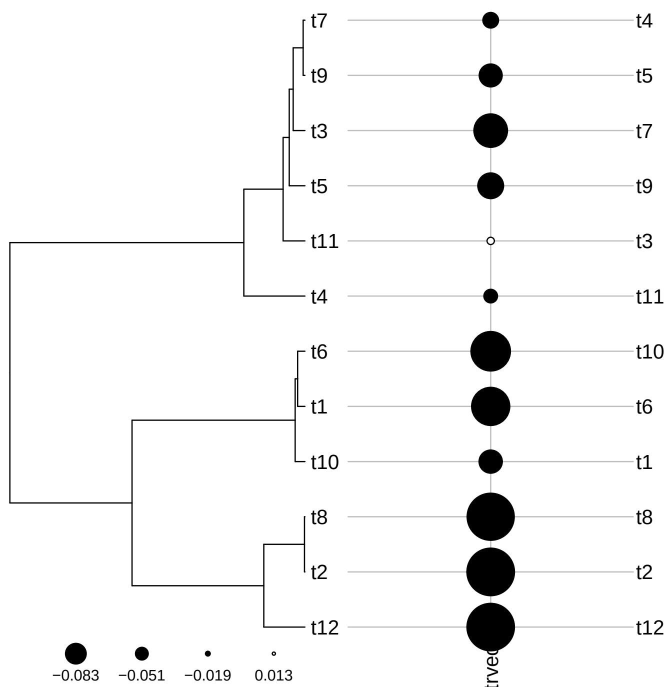
```
> set.seed(1001)
> tree <- as(rcoal(12), "phylo4")
```

Next we generate the phylogenetic variance-covariance matrix (by coercing the tree to a phylo4vcov object) and pick a single set of normally distributed traits (using MASS:mvrnorm to pick a multivariate normal deviate with a variance-covariance matrix that matches the structure of the tree).

```
> vmat <- as(tree, "phylo4vcov")
> vmat <- cov2cor(vmat)
> library(MASS)
> trvec <- mvrnorm(1, mu = rep(0, 12), Sigma = vmat)
```

The last step (easy) is to convert the phylo4vcov object back to a phylo4d object:

```
> treed <- phylo4d(tree, tip.data = as.data.frame(trvec))
> plot(treed)
```



9.1.2 The hard way

```
> ## add node labels so we can match to data
> nodeLabels(tree) <- as.character(sort(nodeId(tree)))
> ## ordering will make sure that we have ancestor value
> ## defined before descendant
> tree <- reorder(tree, "preorder")
```

```
> edgemat <- edges(tree)
> ## set aside space for values
> nodevals <- numeric(nrow(edgemat))
> ## label data in edge matrix order
> names(nodevals) <- labels(tree, "all")[nodeId(tree, "all")]
> ## variance is proportional to edge length; drop first
> ## element of edge length, which is NA
> dvals <- rnorm(nrow(edgemat) - 1, sd=edgeLength(tree)[-1]^2)
> ## indexing: ind[node number] gives position in edge matrix
> ind <- order(nodeId(tree, "all"))
> for (i in 2:nrow(edgemat)) {
+   ## value of ancestor node plus change
+   nodevals[i] <- nodevals[ind[edgemat[i, 1]]] + dvals[i - 1]
+   }
> nodevals <- data.frame(nodevals)
> treed2 <- phylo4d(tree, all.data=nodevals)
```

A Definitions/slots

This section details the internal structure of the `phylo4`, `multiphylo4`, `phylo4d`, and `multiphylo4d` classes. The basic building blocks of these classes are the `phylo4` object and a dataframe. The `phylo4` tree format is largely similar to the one used by `phylo` class in the package `ape` [1].

We use "edge" for ancestor-descendant relationships in the phylogeny (sometimes called "branches") and "edge lengths" for their lengths ("branch lengths"). Most generally, "nodes" are all species in the tree; species with descendants are "internal nodes" (we often refer to these just as "nodes", meaning clear from context); "tips" are species with no descendants. The "root node" is the node with no ancestor (if one exists).

A.1 phylo4

Like `phylo`, the main components of the `phylo4` class are:

edge a 2-column matrix of integers, with N rows for a rooted tree or $N-1$ rows for an unrooted tree and column names `ancestor` and `descendant`. Each row contains information on one edge in the tree. See below for further constraints on the edge matrix.

edge.length numeric list of edge lengths (length N (rooted) or $N-1$ (unrooted) or empty (length 0))

Nnode integer, number of (internal) nodes

[1] http://ape.mpl.ird.fr/

tip.label character vector of tip labels (required), with length=# of tips. Tip labels need not be unique, but data-tree matching with non-unique labels will cause an error

node.label character vector of node labels, length=# of internal nodes or 0 (if empty). Node labels need not be unique, but data-tree matching with non-unique labels will cause an error

order character: "preorder", "postorder", or "unknown" (default), describing the order of rows in the edge matrix. , "pruningwise" and "cladewise" are accepted for compatibility with `ape`

The edge matrix must not contain `NA`s, with the exception of the root node, which has an `NA` for `ancestor`. `phylobase` does not enforce an order on the rows of the edge matrix, but it stores information on the current ordering in the `@order` slot — current allowable values are "unknown" (the default), "preorder" (equivalent to "cladewise" in `ape`) or "postorder": see `http://en.wikipedia.org/wiki/Tree_traversal` for more information on orderings. (`ape`'s "pruningwise" is "bottom-up" ordering.)

The basic criteria for the edge matrix are taken from `ape`, as documented in `ape.mpl.ird.fr/misc/FormatTreeR_28July2008.pdf`. This is a modified version of those rules, for a tree with n tips and m internal nodes:

- Tips (no descendants) are coded $1, \ldots, n$, and internal nodes ($\geq 1 descendant$) are coded $n+1, \ldots, n+m$ ($n+1$ is the root). Both series are numbered with no gaps.

- The first (ancestor) column has only values $> n$ (internal nodes): thus, values $\leq n$ (tips) appear only in the second (descendant) column)

- all internal nodes [not including the root] must appear in the first (ancestor) column at least once [unlike `ape`, which nominally requires each internal node to have at least two descendants (although it doesn't absolutely prohibit them and has a `collapse.singles` function to get rid of them), `phylobase` does allow these "singleton nodes" and has a method `hasSingle` for detecting them]. Singleton nodes can be useful as a way of representing changes along a lineage; they are used this way in the `ouch` package.

- the number of occurrences of a node in the first column is related to the nature of the node: once if it is a singleton, twice if it is dichotomous (i.e., of degree 3 [counting ancestor as well as descendants]), three times if it is trichotomous (degree 4), and so on.

`phylobase` does not technically prohibit reticulations (nodes or tips that appear more than once in the descendant column), but they will probably break most of the methods. Disconnected trees, cycles, and other exotica are not tested for, but will certainly break the methods.

We have defined basic methods for `phylo4`:`show`, `print`, and a variety of accessor functions (see help files). `summary` does not seem to be terribly useful in the context of a "raw" tree, because there is not much to compute.

A.2 phylo4d

The `phylo4d` class extends `phylo4` with data. Tip data, and (internal) node data are stored separately, but can be retrieved together or separately with `tdata(x,"tip")`, `tdata(x,"internal")` or `tdata(x,"all")`. There is no separate slot for edge data, but these can be stored as node data associated with the descendant node.

A.3 multiphylo4